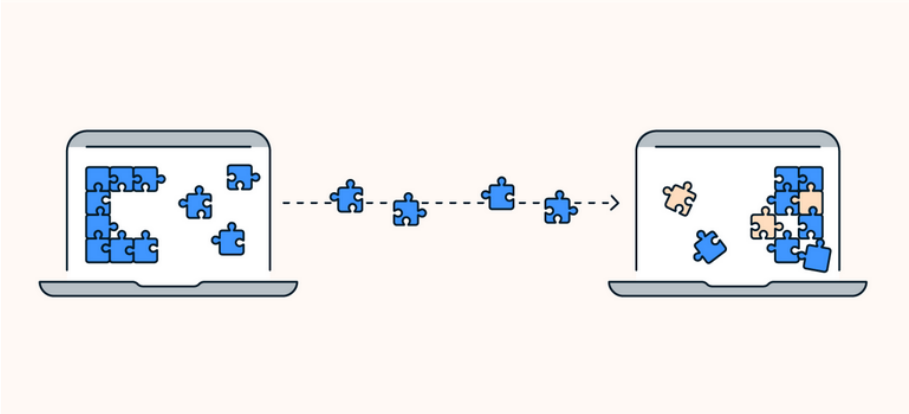
Xarxes de comunicacions

Aplicación Servidor/Cliente para la transmisión de video en tiempo real(streaming)



Autor:

Kevin Brandon Ventura Laura

20 de gener de 2026

1 Introducción

El protocolo estándar de transferencias de archivos entre un ordenador un servidor a través de la red, FTP, se basa en la arquitectura servidor-cliente para transmitir en la red TCP y IP, cuando el cliente solicita un recurso del servidor previamente, siendo el cliente el programa que permite conectarse a un ordenador remoto o servidor, mediante el protocolo FTP.El servidor se ejecuta en un equipo conectado a la red, y recibirá comandos enviados por el cliente, en consecuencia el servidor enviará respuestas y los datos solicitados. A diferencia del protocolo UDP visto en las prácticas, que no establecía conexión previa, y por tanto no había garantias de recibir los datos en orden o la llegada misma, en FTP, se establece conexión previa de cliente-servidor previamente a enviar datos, ya que el protocolo TCP, está orientado a la conexión, una vez establecida la conexión, TCP envia los paquetes de forma ordenada o notifica que se ha producido algún error en caso de fallar.

El programa que se presenta en este miniproyecto, es un cliente-servidor basado en FTP, donde el servidor envía un video de formato mp4 y el cliente solicita ver este video, estableciendo previamente la conexión con el servidor para solicitar el archivo (video), por lo cual, se consigue una aplicación cliente-servidor para la transmisión de video en tiempo real o streaming y a su vez, para medie el rendimiento de transmisión y recepción de video, se calcula FPS para ver su fluidez de transmisión y recepción.

Para este miniproyecto, se ha empleado el lenguaje de programación python y sus extensiones como openCV, socket, etc. Como entorno de desarrollo, se ha utilizado vim en terminal de windows ”powershell”y el compilador de python.

2 Servidor TCP

2.1 Aplicación servidor

La aplicación del servidor escucha por el puerto 9999 y espera que el cliente estaclezca conexión con el servidor para pedir que se le transmita el video. Este servidor esta configurado de manera que solo atiende a un cliente por transmisión, el cliente puede esperar que se termine la transmisión del video, lo que implicaría esperar toda la duración, por lo cual, se ha configurado una tecla para detener la transmisión en cualquier momento (cerrar la conexión), quedando libre para que otro cliente se conecte.

2.2 Código del servidor y funcionamiento

2.2.1 Paquetes de python empleados

```
import cv2, imutils, socket
import numpy as np
import time
import base64
import struct
```

Código 1: Paquetes python empleados

- **cv2**; OpenCV para el procesador del video a transmitir a tiempo real.
- **imutils**: Funciones que permiten realizar el procesamiento de imagenes.

- **socket**: Canal de comunicación entre cliente y servidor, para protocolos TCP o UDP para intercambiar información a través de IPs y puertos de forma bidireccional, por lo cual permite el envío y recepción de datos.
- **numpy**: librería para el procesado de matrices (imágenes).
- **base64**: librería para la conversión de imagen a texto que se pueda enviar.
- **struct**: librería para empaquetar el mensaje en binario.
- **time**: Acceso al tiempo y conversiones para calcular los FPS.

2.2.2 Sockets y configuración de servidor

```
socket_servidor = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

nombre_equipo = socket.gethostname()
ip_servidor = socket.gethostbyname(nombre_equipo)
print('IP del Servidor:', ip_servidor)
puerto = 9999
```

Código 2: Sockets y configuración de red

- Se crea un socket (**socket.socket()**) para comunicación en la red de tipo TCP para IPv4 **socket.socket(socket.AF_INET, socket.SOCK_STREAM)**, donde **socket.AF_INET** especifica que se usa el protocolo IPv4, lo que quiere decir que las conexiones se hacen por direcciones IP de versión 4, por otro lado, **socket.SOCK_STREAM** indica que se crea un socket del tipo Stream, que usa el protocolo TCP, para asegurar que los datos lleguen ordenados y sin sufrir pérdidas.
- **socket.gethostname()**: Obtenemos el nombre del host o red del ordenador donde se ejecutará el código.
- **socket.gethostbyname()**: Se obtiene a partir del nombre del host, la dirección IP correspondiente.
- Se define el puerto 9999 para escuchar las conexiones.

2.2.3 Escuchar clientes

```
# Asociar la IP y puerto y se pone en modo pasivo
socket_servidor.bind((ip_servidor, puerto))
socket_servidor.listen(5)
print(f'Escuchando en la IP :{ip_servidor} y Puerto:{puerto}')

# Espera cliente
socket_cliente, direccion_cliente = socket_servidor.accept()
print('Conexión desde el cliente:', direccion_cliente)
```

Código 3: Espera de conexiones del cliente

- **socket_servidor.bind((ip_servidor, puerto))** : Asignamos la dirección IP del ordenador y el puerto para el socket del servidor.
- **socket_servidor.listen(5)**: Se pone el servidor en modo pasivo, indicando que debe esperar y aceptar la solicitud de conexión de los clientes. En este caso se define 5 para que

el servidor, solo pueda almacenar temporalmente 5 solicitudes de conexión, esto se hace para gestionar mejor el tráfico y evitar sobrecargas.

- **socket_servidor.accept()**: Se espera que se acepte la solicitud de conexión del cliente, y bloquea la ejecución hasta establecer conexión, posteriormente devuelve el socket para comunicarse con el cliente.

2.2.4 Código de envío de video y cerrado de conexiones(socket)

```
# Abrir archivo de video
nombre_video = 'video.mp4'
video = cv2.VideoCapture(nombre_video)
# Variables que nos sirven para calcular el rendimiento (FPS)
fps, tiempo_inicio, frames_para_contar, contador = (0, 0, 40, 0)

try:
    while video.isOpened():
        #Leer los cuadros del video (returno True) y return False si el video se acaba
        ↪  retorno es False
        retorno, cuadro = video.read()
        if not retorno: break
        # Escalamos la imagen para reducir el ancho para que pese menos
        cuadro = imutils.resize(cuadro, width=800)
        #convertimos los cuadros de imagen a formato jpg con una calidad de 80
        codificado, buffer = cv2.imencode('.jpg', cuadro, [cv2.IMWRITE_JPEG_QUALITY, 80])
        mensaje = base64.b64encode(buffer) #convierte los bytes a texto de base64

        #lo primero que hacesmos es ver el tamanyo del mensaje (bytes convertidos en
        ↪  base64),
        #con struck.pack, convertimos el numero en una cadena y posteriormente con
        #socket_cliente.sendall() enviamos el tamanyo del mensaje y el mensaje real
        tam = len(mensaje)
        socket_cliente.sendall(struct.pack(">L", tam) + mensaje)
        # Se dibuja en el recuadro de la imagen emergente el rendimiento (FPS)
        cuadro = cv2.putText(cuadro, 'FPS: '+str(fps), (10, 40), cv2.FONT_HERSHEY_SIMPLEX,
        ↪  0.7, (0, 0, 255), 2)
        cv2.imshow('TRANSMITIENDO', cuadro) #Nombre de pestanya

        # Para salir y romper el bucle, se presiona la tecla "q"
        tecla = cv2.waitKey(1) & 0xFF
        if tecla == ord('q'):
            break

        # calculamos los FPS cada 40 cuadros transmitidos
        if contador == frames_para_contar:
            try:
                fps = round(frames_para_contar/(time.time()-tiempo_inicio))
                tiempo_inicio = time.time()
                contador = 0
            except: pass
        contador += 1
finally:
    # Se cierra los sockets (conexiones) y ventanas al terminar
    socket_cliente.close()
    socket_servidor.close()
    video.release()
    cv2.destroyAllWindows()
```

Código 4: Envío de archivos y cerrado de conexiones al terminar

- **cv2.VideoCapture(nombre_video)** lee el video para enviarlo. Si se pone 0, se está leyendo la camara web.
- **fps, tiempo_inicio, frames_para_contar, contador = (0, 0, 40, 0)** son variables para el cálculo de FPS.

En el cuerpo de **try** se lee el video, con un bucle while que se ejecuta mientras el video este abierto, **video.isOpened()** comprueba que este abierto correctamente, posteriormente, se leen los recuadros del video, escalamos la imagen para reducir el peso con el comando **imutils.resize(cuadro,width=800)**, convertimos los recuadros de imagen a formato jpg con una calidad 80 **cv2.imencode()** y convertimos estos bytes del buffer a formato texto de base64 **base64.b64encode(buffer)**, Con esto hemos convertido, los datos del video en formato texto que si podemos enviar. Para poder enviar primero vemos el tamaño del mensaje (los datos del texto en base64), con **struct.pack** convertimos en tamaño del mensaje (**len(mensaje)**) de número a una cadena y utilizamos **socket_cliente.sendall()** para enviar el tamaño de la cadena y el mensaje real. Además para ver el rendimiento del video, se dibuja un recuadro de texto sobre el video transmitido con **cv2.putText()** que se calcula con el if del contador y se pone nombre a la pestaña con **cv2.imshow()**.

Para romper el bucle, es decir, salir de video, se configura la tecla ”qc̈on **cv2.waitKey() y el registro 0XFF**, finalmente se cierran los sockets y cv2:

- **socket_cliente.close()**:Finalizamos la conexión con el cliente, cerramos el socket.
- **socket_servidor.close()**:Cerramos el socket del servidor.
- **video.release()**: liberamos lo recursos de memoria.
- **cv2.destroyAllWindows()** :Destruye todas las ventanas creadas.

3 Cliente

3.1 Aplicación cliente

La aplicación del cliente, al establecer una conexión con el servidor, permite la recepción del video transmitido por el servidor.

3.2 Código del cliente y funcionamiento

3.2.1 Configuración socket del cliente para usar el servidor TCP

```
import cv2, socket, numpy as np, time, base64, struct

# Configura el socket del cliente para usar TCP
socket_cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
ip_servidor = '192.168.1.20' # IP donde está el servidor (debe coincidir)
puerto = 9999

# Nos conectamos con el servidor
socket_cliente.connect((ip_servidor, puerto))
```

Código 5: Configuración cliente

- Se crea un socket (**socket.socket()**) para comunicación en la red de tipo TCP para IPv4 **socket.socket(socket.AF_INET, socket.SOCK_STREAM)**, donde **socket.AF_INET** especifica que se usa el protocolo IPv4, lo que quiere decir que las conexiones se hacen por direcciones IP de versión 4, por otro lado, **socket.SOCK_STREAM** indica que se crea un socket del tipo Stream, que usa el protocolo TCP, para asegurar que los datos lleguen ordenados y sin sufrir pérdidas.

- **ip_servidor** es la IP donde está configurado nuestro servidor junto a su **puerto**.
- **socket_cliente.connect()**: Nos conectamos con el servidor mediante su IP y puerto.

3.2.2 Código de recepción de video y cerrado de sockets

```
datos_recibidos = b"" # Buffer para acumular los datos
tam_encabezado = struct.calcsize(">L") # El encabezado debe medir 4 bytes
# Variables que nos sirven para calcular el rendimiento (FPS)
fps, tiempo_inicio, frames_para_contar, contador = (0, 0, 40, 0)

while True:
    # Recibimos los primero 4 bytes para saber de que tamanyo es la imagen a recibir
    while len(datos_recibidos) < tam_encabezado:
        paquete = socket_cliente.recv(4096) #leemos los 4096 bytes, paquetes
        if not paquete: break
        datos_recibidos += paquete

    if not datos_recibidos: break

    # Extraemos el tamanyo de lo recibido y se limpia el buffer de esos 4 bytes recibidos
    encabezado_empaquetado = datos_recibidos[:tam_encabezado]
    datos_recibidos = datos_recibidos[tam_encabezado:]
    tam_mensaje = struct.unpack(">L", encabezado_empaquetado)[0]

    # continuamos recibiendo los datos hasta completar el tamanyo de datos de la imagen
    while len(datos_recibidos) < tam_mensaje:
        datos_recibidos += socket_cliente.recv(4096)#leemos los 4096 bytes

    # Se extrae los datos de cada cuadro y se guarda el resto para la siguiente iteracion
    datos_cuadro = datos_recibidos[:tam_mensaje]
    datos_recibidos = datos_recibidos[tam_mensaje:]

    #reconstruccion de imagen
    #Pasamos de la base64 a bytes, y de bytes a una matriz numpy de tipo uint.8
    #posteriormente se pasa a imagen con OpenCV para reconstruir el cuadro
    imagen_binaria = base64.b64decode(datos_cuadro)
    datos_numpy = np.frombuffer(imagen_binaria, dtype=np.uint8)
    cuadro = cv2.imdecode(datos_numpy, 1)

    # Se dibuja en el recuadro de la imagen emergente el rendimiento (FPS)
    cuadro = cv2.putText(cuadro, 'FPS: '+str(fps), (10, 40), cv2.FONT_HERSHEY_SIMPLEX,
    ↪  0.7, (0, 0, 255), 2)
    cv2.imshow("RECIBIDO", cuadro) #nombre de pestanya

    # Para salir y romper el bucle, se presiona la tecla "q"
    tecla = cv2.waitKey(1) & 0xFF
    if tecla == ord('q'):
        break

    # calculamos los FPS cada 40 cuadros transmitidos
    if contador == frames_para_contar:
        try:
            fps = round(frames_para_contar/(time.time()-tiempo_inicio))
            tiempo_inicio = time.time()
            contador = 0
        except: pass
    contador += 1

socket_cliente.close()
cv2.destroyAllWindows()
```

Código 6: Recepción de archivos y cerrado de sockets al terminar

- **datos_recibidos**: variable para el buffer acumulador de datos.
- **struct.calcsize()**: Se calcula el tamaño en byte lo que ocuparán los datos binarios.

- **fps, tiempo_inicio, frames_para_contar, contador = (0, 0, 40, 0)** son variables para el cálculo de FPS.

El cuerpo del código principal **while True**, se recibe mientras se transmita los datos, el siguiente **while** nos sirve para calcular, el tamaño de la imagen a recibir después de recibir los 4 byte primeros y se leen los datos recibidos con **socket_cliente.recv()** si estos datos no estan en el paquete, se sale del bucle, por el contrario, se guardan en el buffer **datos_recibidos**. Una vez extraído el tamaño de lo recibido, se limpia el buffer y se convierten los bytes los datos binarios de **datos_recibidos**. Posteriormente con el siguiente **while** se sigue recibiendo los datos **socket_cliente.recv()** hasta completar el tamaño de la imagen, guardamos en el buffer lo leído y se extrae los datos de cada cuadro. Finalmente para reconstruir la imagen, se pasa de datos binarios base64 a bytes, con **base64.b64decode()** (imagen binaria), de bytes a una matriz de numpy del tipo uint.8 **np.frombuffer(imagen_binaria, dtype=np.uint8)** y de la matriz numpy se pasa a reconstruir el cuadro con la librería openCV **cuadro = cv2.imdecode(datos_numpy, 1)**, por lo cual, ya tenemos el cuadro reconstruido. Además como hemos hecho en el caso del servidor, calculamos los FPS del video recibido para ver el rendimiento, configuramos una tecla de salida y cerramos los sockets del cliente **socket_cliente.close()** y destruimos la ventana emergente creada con **cv2.destroyAllWindows()**.

4 Resultados

Una vez tenemos los códigos terminados del cliente-servidor en una carpeta, además del video a transmitir (el video se puede tener en otra carpeta, pero se tendría que especificar la ruta absoluta), abrimos 2 terminales, una para el servidor y otra para el cliente, compilamos código del servidor con **python servidor.py**, por lo cual, la terminal nos muestra en la figura 1 que se encuentra en espera.

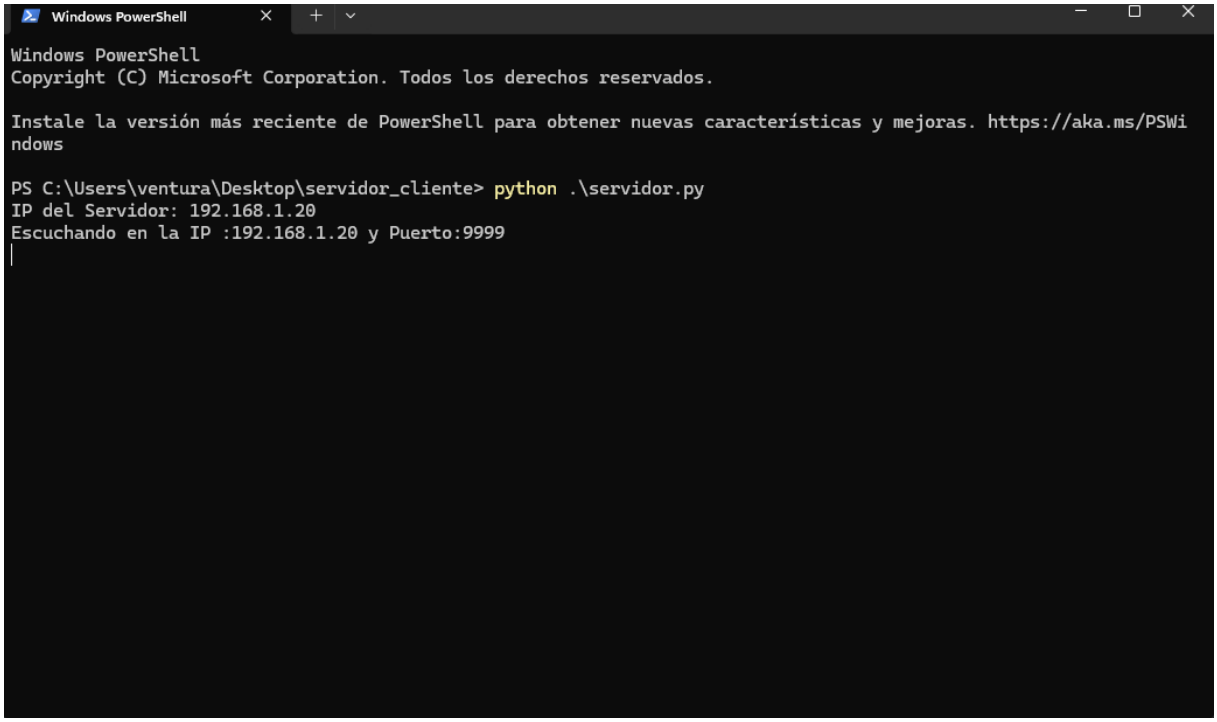


Figura 1: Servidor en espera

Conectamos el cliente con el servidor que en espera, compilando el código cliente.py con **python cliente.py** y vemos en la figura 2 que el cliente se ha conectado al servidor con éxito, por tanto, el cliente esta recibiendo lo demandado al servidor.

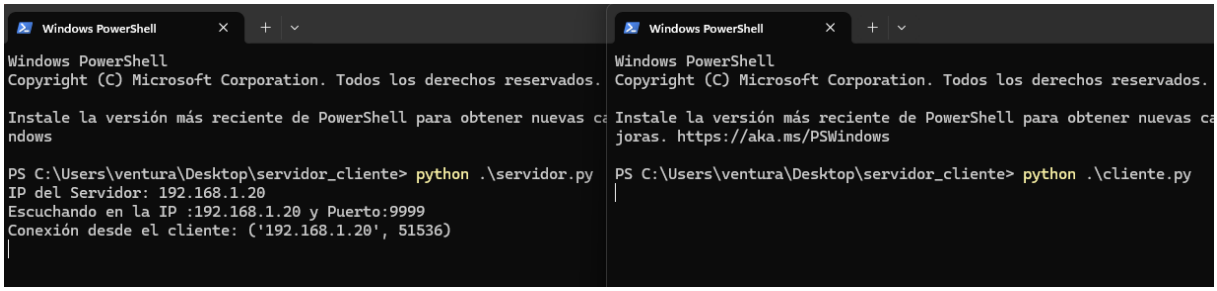


Figura 2: Conexión Cliente-Servidor

En la figura 3 se puede ver que al establecer la conexión entre servidor y cliente, se abren dos pestañas emergentes, una que se trata del video que es transmitido por el servidor y otra donde el cliente recibe el video, además que en cada recuadro, se puede observar el rendimiento de transmisión y recepción.

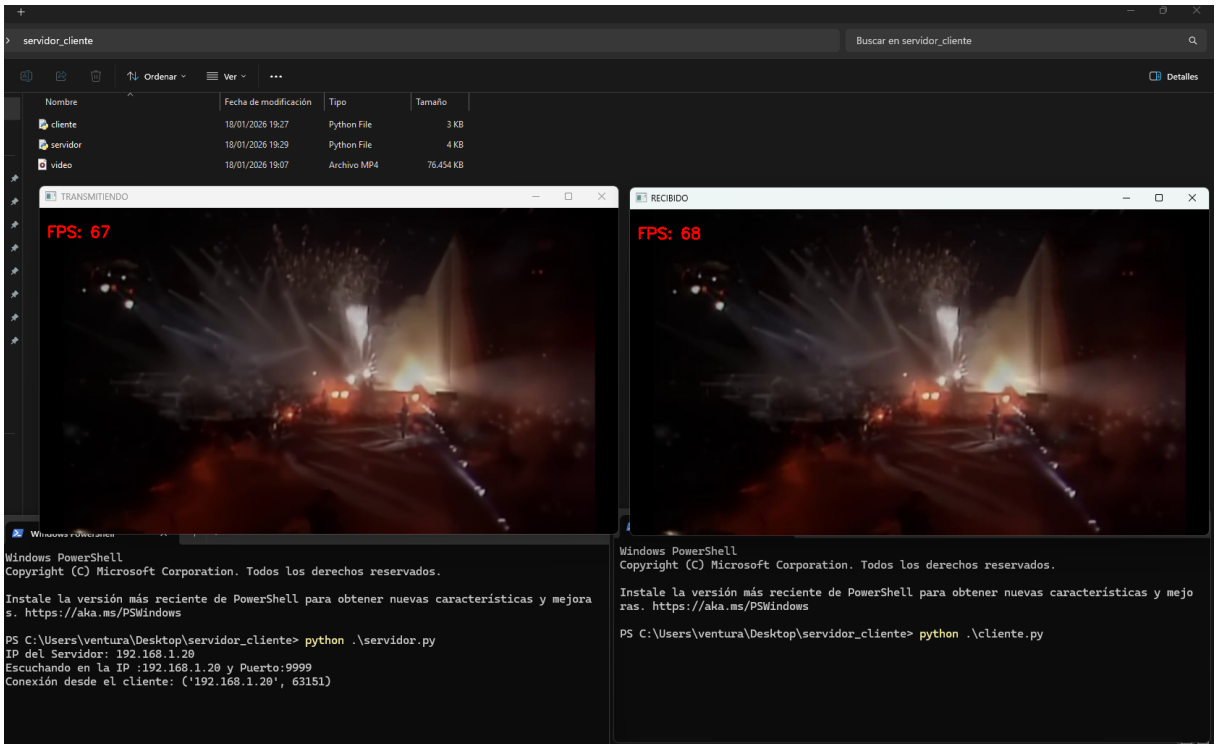


Figura 3: Demostración de transmisión y recepción de video a tiempo real

5 Conclusiones

En conclusión, se implemento con éxito la aplicación cliente-servidor TCP para la transmisión de video a tiempo real entre el servidor y el cliente con un rendimiento bastante bueno, con una comunicación de TCP que nos garantizó la llegada de datos ordenados sin pérdidas, por lo cual el video se reprodujo correctamente en el cliente, con un rendimiento bueno, comparando

el rendimiento entre el video emitido y recibido. Este miniproyecto, por tanto, demuestra la viabilidad de crear un sistema de streaming básico usando python y OpenCV de licencia abierta.